2015 IEEE International Parallel and Distributed Processing Symposium Workshop

High-Performance Serverless Data Transfer over Wide-Area Networks

Eun-Sung Jung, Rajkumar Kettimuthu
Mathematics and Computer Science Division
Argonne National Laboratory
Email: {esjung,kettimut}@mcs.anl.gov

***Abstract*—We propose a serverless data movement architecture in which we bypass data transfer nodes, the filesystem stack, and the host system stack and directly move the data from one disk array controller to another in order to obtain the highest end-to-end performance. Under the current data movement architecture, separate data transfer nodes arbitrate data transfer by input/output to parallel file systems over local-area networks and parallel file systems read/write actual data from/to disks through disk controllers. Our new architecture embeds parallel file system servers and data transfer processes into a disk controller to eliminate the data path between a data transfer process and a parallel file system. This prevents the network between those two entities from being a bottleneck in end-to-end data transfer. In addition, we propose a parallel data-layout-aware data transfer, where multiple embedded data transfer processes send segments of a file concurrently while considering data layout in disks to improve data transfer performance. Our experimental results show that our proposed architecture is feasible and outperforms the traditional architecture significantly.**

I. Introduction

The data produced at U.S. Department of Energy (DOE) experimental and computing facilities is often shared among the facilities and collaborating institutions, and the volume of this data is expected to increase exponentially in the next several years. Moving large datasets rapidly over wide-area networks for remote analysis, data distribution, and replication is becoming a key requirement for science. The emerging extreme-scale science era will see large numbers of high-performance data flows with varying quality-of-service (QoS) requirements. These data flows can be four types: memory to memory (M2M), memory to disk (M2D), disk to memory (D2M), and disk to disk (D2D). Science workflows will have data transfers with computer or instrument memory as source and/or computer memory as destination as well as with disk as source and/or destination, since the memory capacities of the computer systems are not growing at the same rate as the data acquisition by the sequencers and detectors [1].

Next-generation terabit networks will help. However, the parallel storage systems on the end-system hosts at institutions can become a bottleneck for terabit data movement. Current storage systems are massively parallel rather than a single disk and are front ended by a complex parallel file system (PFS) stack. PFS is a widely adopted solution for scientific applications to support both high performance I/O and large data sets. PFS utilizes many I/O servers each with many disks for performance and capacity scaling and stores individual files over subsets of disks to improve single file performance.

In this paper, our contribution is threefold. We propose a novel serverless data transfer architecture that eliminates a long I/O path caused by dedicated data transfer nodes (DTNs). We also propose an efficient parallel data transfer mechanism using data layout information. We provide a practical evaluation by applying our new methods to actual hardware and software packages.

The rest of the paper is structured as follows. In Section II we present motivations, derived from the drawbacks of existing systems, for our work. In Section III we present a novel data transfer architecture, and in Section IV we present a data layout-aware parallel data transfer mechanism. In Section V we discuss how we can deploy our methods in a real disk controller with the capacity of running a few virtual machines. In Section VI we present experimental results evaluating our proposed methods. In Section VII we describe related work in detail. In Section VIII we summarize our work and conclude with ideas for future work.

II. Motivation

Typically, PFS is shared by compute clusters and DTNs—basically, dedicated computers used to switch between network and disks [2]. Figure 1 shows the interaction between the DTN and PFS for I/O requests. For an I/O operation such as a read or write, the request is processed by several layers of system software such as file system, logical device drivers, and hardware device drivers. A single application I/O request may be split into several "physical" I/O requests, and these requests must pass through the interconnection switch that connects the host bus adapter on the computer system and the disk array controller. The disk array controller takes the I/O request and creates one or more I/O requests to the individual disks in the array. The data is then transferred between the disk drive and the application memory space with multiple buffering along the way. Obtaining the highest performance into and out of such complex I/O systems can be a complicated process, involving significant software development and tuning.

Although existing data movement efforts have been successful in pushing file transfer performance above 10 Gbps, they will not scale well to 100 Gbps and 1 Tbps. We need innovative approaches and new data movement architectures to push end-to-end data transfer performance to 1 Tbps. In the

978-1-4673-7684-6 2015

DOI 10.1109/IPDPSW.2015.69

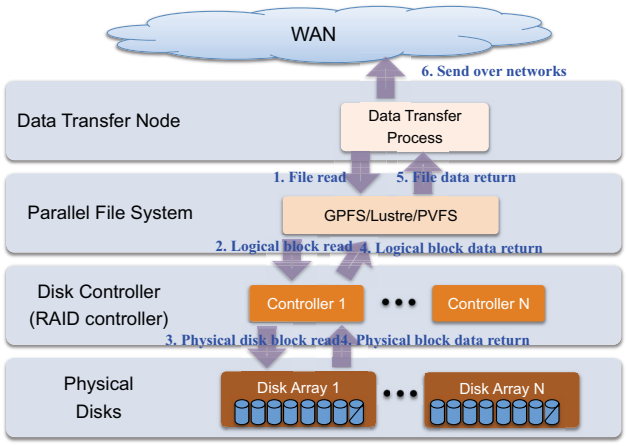

Fig. 1: File read process in the current data transfer architecture.

early Internet, computers were used as routers (e.g., ARPAnet IMP, and later systems), which turned out to be inefficient. But for wide-area data transfers, the best practice is still to use dedicated servers (DTNs). When networks run at terabits per second and a large number of high-performance data flows emerges, we cannot have these servers in the way between network and disks. The data-intensive computer architecture is expected to evolve to accommodate higher-speed data rates in all levels of data management [1]. For example, the storage on the node is projected to be 10–100 TB SSD cache or local filesystem; and high-speed network interface cards (i.e., 40 GB ethernet) will be installed in each node. Even with higher capacity hardware in I/O paths, however, the complex and deep I/O systems as in Fig. 1 still pose challenges, as follows.

Deep I/O Path Despite Direct I/O: The remote direct memory access (RDMA) for wide-area data transfer has been used on reserved network paths. It avoids kernel memory copies by bypassing the operating system. The overhead in end-to-end data transfers in high-performance computing systems nevertheless remains nontrivial because of deep I/O paths spanning from disks to hosts via PFS.

Shared Parallel Filesystems: The large PFS used by big computing facilities is designed to produce large amounts of aggregate storage bandwidth and provide diverse parallel access semantics. One primary issue is how applications can achieve required disk throughput. If 1,000 processes are doing file system I/O, each process needs to get only 125 MB/s to achieve an aggregate throughput of 125 GB/s (1 Tbps). In the case of a file transfer application using a handful of data transfer nodes, a single-process I/O performance of 125 MB/s will be insufficient to saturate even a 10 Gbps network link.

Unpredictable I/O Performance: The deep I/O path also prevents applications from predicting the I/O performance within a guaranteed range of variance. For example, if disks are shared by multiple applications, it is hard to guarantee and/or predict disk I/O performance. With deeper I/O systems as in Fig. 1, it is intractable to guarantee and/or predict I/O performance because of resource contention by many applications, which offset the advantages of bandwidth-guaranteed network paths through software-defined networks coming in the near future.

III. Serverless Data Transfer

We propose a serverless data movement architecture, shown in Fig. 2, in which we bypass DTNs, the filesystem stack, and the host system stack and directly move the data from one disk array controller to another in order to obtain the highest end-to-end performance. Instead of a DTN with a data transfer process (DTP), a control DTP, which can be placed anywhere, negotiates with PFS to get real block locations related to the data. The control DTP then arbitrates data transfer with data mover DTPs in a disk controller.

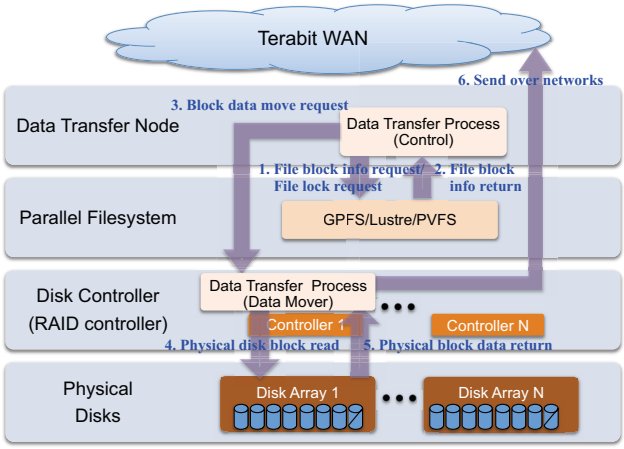

Fig. 2: Serverless data transfer architecture.

Few commercial products such as DataDirect Networks (DDN) SFA12K [3] has a similar architecture, in where a controller has enough hardware capacity (e.g., several CPUs) to accommodate virtual machines with embedded parallel file system servers. Fig. 3 shows the reduced I/O path of DDN SFA12K with embedded parallel file system servers where we can eliminate the communication overhead between parallel file system servers and disk controllers.

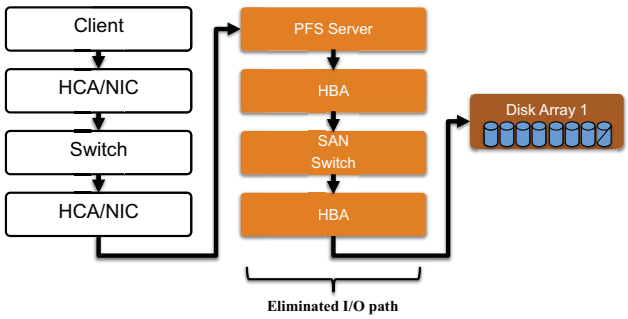

Fig. 3: Reduced I/O path in DDN SFA12K.

Our serverless data movement architecture goes further than the architecture in Fig. 3. It can eliminate the I/O path from a client to a server since a client, a data transfer process, is also

placed in the same disk controller where a PFS server runs. For the rest of this paper, we will refer to this serverless data movement architecture as the embedded DTP architecture as opposed to the traditional DTN architecture.

IV. Layout-Aware Parallel Data Transfer

In general, a data transfer process does not know where the data blocks for a file are located across actual disks and does not know how many parallel file servers are involved in reads and writes of a file. For example, if a remote client read a large size file from a parallel file system consisting of 8 servers with 10 disks, it is transparent to the client that 8 servers read fragments of the file from 10 disks concurrently and send the data over the network.

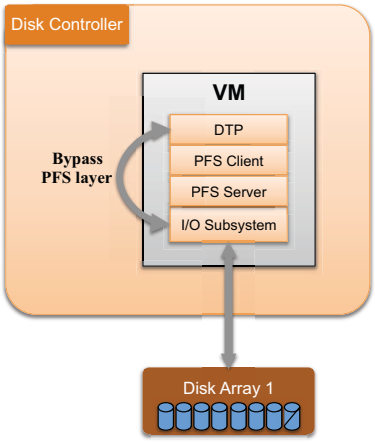

Fig. 4: Direct I/O access to disks bypassing PFS layer.

This situation changes in two aspects if a data transfer process is embedded into the disk controller together with parallel file servers. First, the data transfer process reads/writes to/from the parallel file system not over networks but through local memory access. This approach eliminates any chance that the network between a data transfer process and a storage system (i.e., a parallel file system) becomes a bottleneck in an end-to-end data transfer procedure. Second, the data transfer process is able to see how many parallel file servers and disks exist in a parallel file system. This approach gives us a new opportunity to shorten the I/O path and fully exploit multiple disk I/O in parallel. As long as we know data layout information of a file, we don't need a file system layer to pull data from disks, as in Fig. 4. Here, we assume that both DTP and PFS are embedded into a virtual machine (VM) hosted by a disk controller. In addition, we can improve the data I/O performance by assigning separate disk groups to different data transfer processes based on data layout information on the fly so that the data I/O degradation caused by multiple intervening processes can be prevented, as in Fig. 5. Note that one embedded data transfer process in a certain host/VM may activate only one parallel file server in the same host/VM because of PFS's own operation policy. This is shown by our experiments, and we thus argue that parallel data transfer processes doing parallel disk I/O are preferred for the embedded data transfer process architectures.

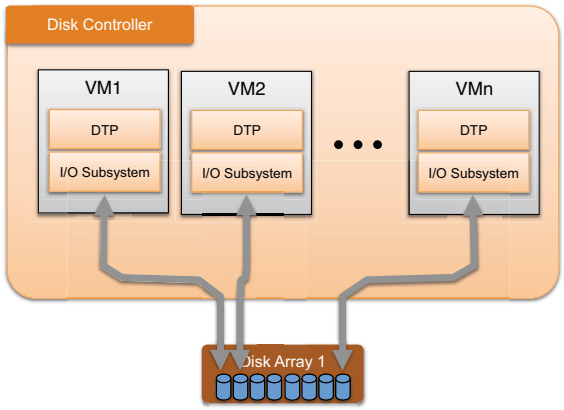

Fig. 5: Parallel disk I/O access in the embedded DTP architecture.

To utilize data layout information of a file, however we may need root privilege to access such information from PFS. In case of Lustre PFS, there is a user API that provides locations of objects (i.e., data blocks) of a file. In case of GPFS, we, however, can get data layout information of a file only with root privilege using the GPFS API hidden from normal users. An example of such information is shown in Listing 1. Here, the FS block size of a GPFS is 4 MB, and file data is striped across the disks which the GPFS manages. We thus locate where a certain 4 MB segment of a file is placed in a disk using a two-tuple (device name, sector number). Note that the overhead of getting information on data blocks is also existing in typical data transfer on top of PFS; To read a file, PFS reads metadata of the file, and reads the real data blocks of the file.

Listing 1: Example of data layout information of a file

```
FS Block Size: 4194304
Inode: 954918          Size: 0x1900000000 B

start offset - end offset    NSD:sector
(in bytes)     (in bytes)
-------------------------------------------------
0 - 0x3fffff                 DDN7_14:14694146048
0x400000 - 0x7fffff          DDN7_15:14206664704
0x800000 - 0xbfffff          DDN7_16:1601789952
0xc00000 - 0xffffff          DDN7_17:4039196672
0x1000000 - 0x13fffff        DDN7_18:7242645504
0x1400000 - 0x17fffff        DDN7_19:1741070336
0x1800000 - 0x1bfffff        DDN7_20:8356888576
0x1c00000 - 0x1ffffff        DDN7_21:3760635904
0x2000000 - 0x23fffff        DDN7_22:4039196672
0x2400000 - 0x27fffff        DDN7_23:13301342208
0x2800000 - 0x2bfffff        DDN7_24:2437472256
-------------------------------------------------
```

On a receiver side, we need also some mechanisms to reserve space for incoming data blocks at proper locations. Even though such mechanisms are not implemented in this paper, we believe it can be done at several levels such as file system specific utilities and MPI API [4]. The most feasible method is to use *fallocate* file system call in UNIX/Linux systems [4]. The system call preallocates blocks to a file and ensures that the space is allocated to the file really not virtually. The more performance may be achieved through temporary high-speed storage devices such as SSDs for incoming data,

and sequential writes similar to log-structured file systems [5], which will be written to real PFS later. Our future work will investigate these options.

V. Putting it all together

We can summarize the whole file transfer procedure in the new embedded DTP architecture as follows.

1) DTPs at the sender and the receiver sites get notified of a file transfer request.
2) The DTPs at the sender site get data layout information from the PFS servers in the same node.
3) The proper number of DTPs at the receiver site based on the architecture is launched.
4) The DTPs at the sender site send the segments of a file concurrently, and each DTP is responsible of different groups of disks.

If a disk controller is equipped with several cores and is able to accommodate several VMs, we can either deploy PFS servers and DTPs on a VM with multiple virtual CPUs or deploy them on separate VMs such that data layout information exchange is doable over high-speed local networks. For example, DDN SFA12KE [3] has two disk controllers and can support up to eight VMs, four VMs per controller.

We use the Globus GridFTP toolkit [6] to implement DTPs, in particular, *globus-url-copy* to initiate data transfers. To send segments of a files, we use the *-off* and *-len* options, which specify the offset and the length of a segment of a file, repeatedly, in *globus-url-copy*.

VI. Experimental Evaluation

In this section, we first present the testbed configuration for the evaluation of our novel hardware and software architecture and then give detailed benchmarks comparing the existing architecture and our architecture in various aspects.

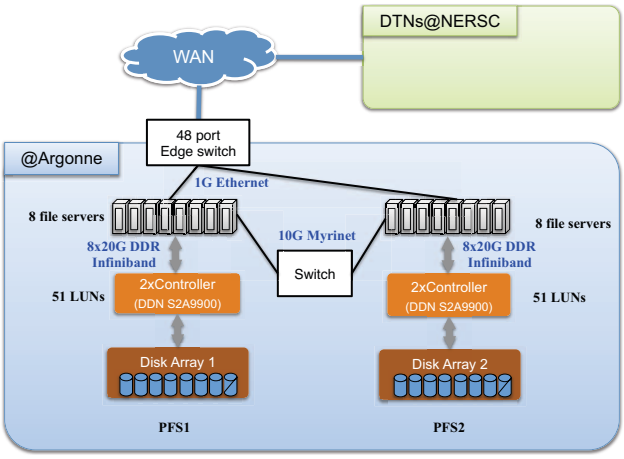

Fig. 6: Testbed configuration.

A. Testbed configuration

We conducted experiments on the testbed located at two sites, Argonne (IL) and NERSC (CA), as shown in Fig. 6. At Argonne, two sets of identical parallel file systems, PFS1 and PFS2, were set up. Each set consists of a disk array, a pair of disk controllers, and 8 physical machines in which parallel file servers are running. In our experiments, IBM GPFS is used as a parallel file system. Each controller exports 51 LUNs to physical machines and has four 20 Gb DDR InfiniBand ports, each of which is connected to one physical machine. Therefore, we can assume that this configuration represents the disk controller with embedded file servers because the network bandwidth between the disk disk controller is very high (20 Gb). The physical machines in PFS1 are connected to the ones in PFS2 through 10 Gb Myrinet such that one machine in PFS1 can mount the PFS2 file system and vice versa. The physical machines are connected to a 48-port edge switch via 1 Gb NICs and can send data to other machines at NERSC. The system architecture at NERSC is a typical architecture consisting of 4 DTNs and parallel file systems accessible by DTNs and other computing nodes. Due to limitations of network connectivity between two sites, most of experiments are simulated at the Argonne site.

B. Baseline experiments

We measured the baseline network throughput between Argonne and NERSC by running *iperf* in one node on each side. The results are not symmetric. Specifically, the network throughput from Argonne to NERSC is about 940 Mbps, whereas the network throughput from NERSC to Argonne is about 105 Mbps. In our testbed, the network throughput between two hosts at Argonne and NERSC is limited by 1 GbE NIC installed on the node at Argonne. Since the network throughput from NERSC to Argonne is too small for some other reasons such as network congestion on the path, we did our experiments on the unidirectional case from Argonne to NERSC.

We conducted experiments on various file sizes from 100 KB to 10 GB while keeping the total transferred data size the same as 10 GB to measure the effects of file size on the performance. For example, with the 1 GB file size, ten 1 GB files are transferred.

Fig. 7 shows the read/write performance of parallel file systems at both Argonne and NERSC. The read/write time for a certain file size is averaged over three runs. The results in Fig. 7 show that the time taken for reads and writes tends to increase as the file size decreases. Note that the NERSC system is a production system where multiple users are actually doing data transfers and the measured times may not reflect accurate system performance. These results are obvious in that the ratio of the metadata to the real data of a file increases as the file size decreases; moreover, small files tend not to be allocated consecutively in a disk, thus slow read/write operations on disks. The write performance is generally better than the read performance in GPFS because, from an application's point of view, a write operation is

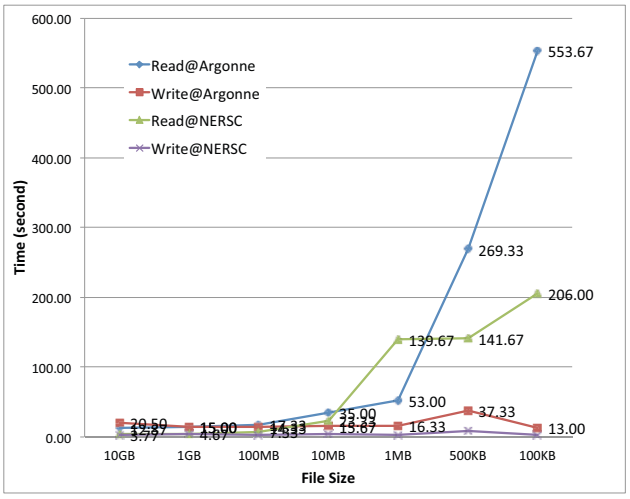

Fig. 7: Parallel file system performance at Argonne and NERSC.

completed as soon as an application's write buffer has been copied to the pagepool of GPFS [7].

C. Separate data transfer node vs. embedded data transfer process

We compare the performance of the embedded data transfer process architecture with the traditional data transfer node architecture. To simulate the traditional DTN architecture, we run Globus GridFTP servers on the hosts of PFS1 and read files stored on disk array 2 through file servers of PFS2, as in Fig. 8. To simulate the embedded DTP architecture, we run Globus GridFTP servers on the hosts of PFS1 together with file servers and read files stored on disk array 1, as in Fig. 9.

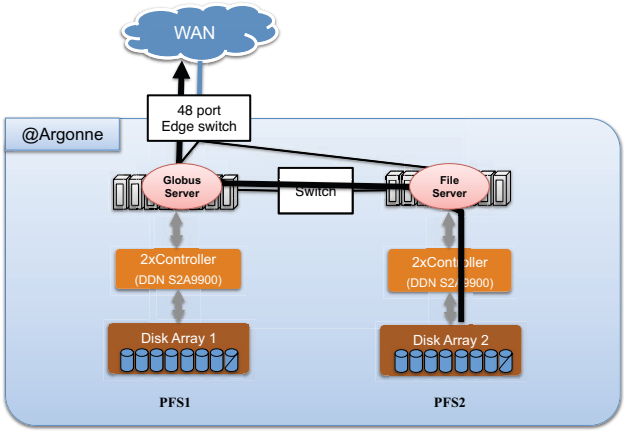

Fig. 8: Simulation of the traditional DTN architecture.

The Globus GridFTP provides several parameters for the tuning of file transfer performance. Among them, three parameters are most influential on the performance [8]: (1) the number of concurrency (-cc) which specifies how many servers are spawned for the transfer and is essential for lots of small files; (2) the number of streams (-p) which specifies how many

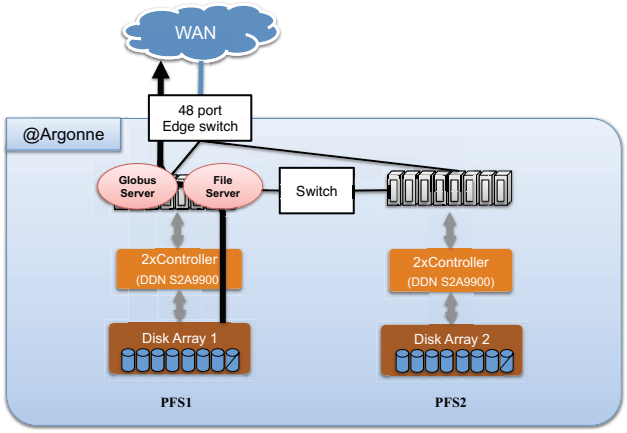

Fig. 9: Simulation of the embedded DTP architecture.

partitions for a file are streamed concurrently to the destination and is essential for a large file; and (3) pipelining (-pp) which enables pipelined control message exchange to hide message exchange latency in the case of lots of small files. We use the following parameters for datasets of different file sizes: -p 4 for 10 GB, -p 4 -cc 2 -pp for 1 GB, 100 MB, and 10 MB, -p 1 -cc 4 -pp for 1 MB, and -p 1 -cc 16 -pp for 100 KB.

1) Data transfer from Argonne to NERSC: We first performed experiments regarding data transfer from Argonne to NERSC using a single host. The results in Fig. 10 show not much difference between the traditional DTN architecture and the embedded DTP architecture except for a small gap in the case of 100 KB file. That result is because the overall performance is limited by the capacity (1 Gbps) of the NIC connected to WAN.

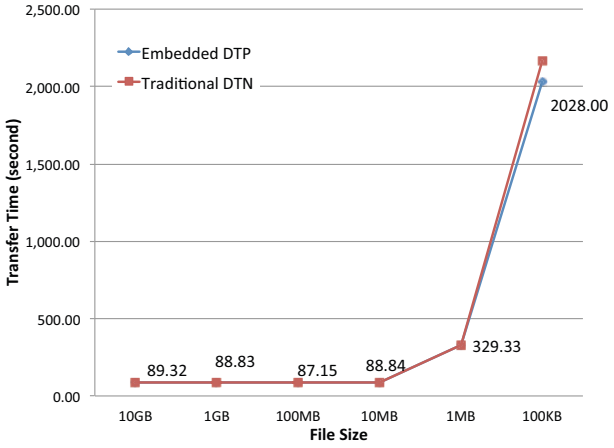

Fig. 10: Performance comparison in WAN data transfer from Argonne to NERSC.

2) Data transfer from Argonne to Argonne – Removing NIC bottleneck: To simulate the situation in which the capacity of NIC is not bottleneck anymore, we ran an additional Globus GridFTP server on a different port. We then performed data transfer locally between two GridFTP servers and set the

destination files to */dev/null* such that no overhead is incurred on the receiver side.

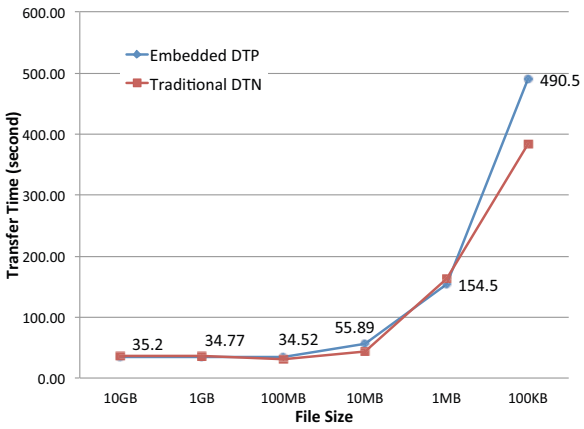

Fig. 11: Performance comparison when the transfer is locally redirected to */dev/null* at Argonne *(unoptimized embedded DTP)*.

These experiments in Fig. 11 result in improved performance compared with the previous experiment in Fig. 10. But, in these experiments, the performance of the embedded DTP architecture is worse than the performance of the traditional DTN architecture. To further analyze the reasons for such results, we performed microbenchmarks.

a) Improving disk I/O of the embedded DTP architecture: We observed that the disk I/O performance in the embedded DTP architecture is inferior to the performance in the traditional DTN architecture. The reason is that the I/O operation in the same node where PFS exists activates only one PFS server, whereas the I/O operation from the remote client activates all the PFS servers (i.e., 8 servers in our testbed). One process cannot fully exploit the maximum throughput of 51 disks in our testbed. We thus run 4 *globus-url-copy* processes to concurrently read and send evenly partitioned regions of a file. In this way, we improve the performance by almost 100%, 17.9 s for 10 GB transfer, as in Table I.

b) Removing any overheads except PFS: We suspect that the local data transfer between two Globus GridFTP servers may also be bottleneck. To remove any bottleneck except PFS related operations, we redirect the destination directly to */dev/null* not to */dev/null* via a GridFTP server. For 10 GB, the results are 11.5 s and 16.51 s for the embedded DTP architecture and the DTN architecture, respectively. In other words, the embedded DTP architecture outperforms the DTN architecture by 43%.

c) Overall comparison: For data transfer using single node, the embedded DTP architecture could achieve performance improvement up to about 100% using optimized disk I/O and shortened I/O path eliminating the network communication between a DTP and PFS servers. Table I shows a summary of data transfer performances between the DTN architecture and the embedded DTP architecture regarding three cases. In the case of FTP-FTP, a file is transferred from one Globus GridFTP server to the other Globus GridFTP server which simply redirects the file to */dev/null*. In the case of FTP-File, a file is read from one Globus GridFTP server and written to */dev/null* without going through a Globus GridFTP server on the receiver side. In the case of File-File, a file is read from a command (i.e., *globus-url-copy*) and written to */dev/null*. The performance improvement may come either from low latency of the shortened I/O path or being free of network bottleneck between a DTP and PFS servers.

TABLE I: Data transfer performance comparison in single-node tests.

Source-Destination Type	Embedded DTP	DTN	Performance Improvement
FTP-FTP	17.94	36.55	103%
FTP-File	17.82	37.49	110%
File-File	11.50	16.51	43%

TABLE II: Performance comparison with regard to data layout aware data transfer

Architecture	Transfer Time	Improvement
Normal embedded DTP	57	Baseline
Layout aware DTP	47	21%
Layout aware DTP w/ sorted segments	40	42%

D. Data-layout-aware transfer using multiple hosts

In addition, we conducted experiments involving data-layout-aware transfer as described in Section IV. We compare the following two cases. In the first case, the embedded DTP architecture uses I/O through PFS (DTP-PFS): one DTP at each of 8 hosts concurrently one of 8 partitions of a 100 GB file by reading from a PFS directory. For example, the first node is responsible for sending a partition ranging from 0 to 12.5 GB in the logical address of the file. In the second case, the embedded DTP architecture uses I/O through raw devices (DTP-RAW): one DTP at each of 8 hosts is assigned to a different group of disks and sends segments of a file belonging to the assigned group of disks. In our testbed, there are 51 LUNs and one host is assigned to 6-7 disks. For example, the first host is assigned to */dev/sdc* through */dev/sdh*.

We coded a script for the DTP-RAW case in python. As many threads as assigned disks are launched at one host to fully exploit disk I/O throughput. Each thread repeatedly runs *globus-url-copy -off offset -len 4 MB* for a segment because the file block size in our testbed is 4 MB. We have a master node in which we remotely start the script in each of 8 hosts using the *ssh* command. We flush the disk cache ahead of each run in order to exclude cache effects. We do not use any DTP server (i.e., *globus-gridftp-server*) since repeated calls of *globus-url-copy* in the DTP-RAW cause more overhead in terms of setting up data and control channels and hence may lead to unfair comparison. We copy from a file and redirect the destination of a file to */dev/null* at each host such that no DTP server is involved in order to measure the PFS overhead accurately. Fortunately, *globus-url-copy* supports the file input with regard to multiple file segment transfers such that the binary executable does not have to be initiated repeatedly.

We observed I/O activities for all the disks at each node in the DTP-PFS case since the DTP at each node sends a

TABLE III: Data flow classification

Class	Representative Example	Benefits from New Approach
M2M	In-Situ Analysis and Visualization (e.g., Cosmology simulations)	Data in memory should eventually be stored in permanent storage. Faster background M2D transfers – rates comparable to M2M transfers that happen in parallel to avoid discarding data.
M2D	Data Gathering and Analysis from Experimental Facilities (e.g., High Energy Physics, Light Sources)	Faster transfer of data from experimental facilities to user site or remote postprocessing facility: Move data from acquisition machine's memory directly to the disk at remote site.
D2M	Remote Data Retrieval for Post Processing (e.g., Light Sources)	Similar way as described in the case of M2D.
D2D	Data Replication for Efficient Distribution (e.g., Climate Science)	Ultra high-speed replication through data transfers directly from disk controllers on one end to another.

partition of a 100 GB file spanning multiple disks. In contrast, we observed I/O activities for assigned disks at each node in the DTP-RAW case. To improve the disk I/O throughput, we sort the order of file segments by the sector numbers of the file segments such that sequential reads rather than random reads are conducted on physical disks. The results are shown in Table II. The data-layout-aware data transfer method can achieve up to 42% better performance than that of the normal embedded DTP architecture.

VII. Related Work

Similar work has been done in the context of backup and high-performance computing. A network data management protocol [9] has been proposed to achieve higher disk backup performance through direct data transfers between backup devices. A high-performance storage system (HPSS) [10] has been developed by IBM with DOE laboratories for extreme-scale storages. HPSS has a cluster design where data mover hosts are dedicated to data transfers between tape drives and disks, and data transfers from HPSS to clients. We note that our approach differs significantly from such work in that we focus more on the least overhead on data transfers without dedicated hosts, guaranteeing varying QoS of data flows, and large numbers of data transfers of various types such as M2M, M2D, D2M, and D2D. Table III shows how our approach benefits a broad range of scientific applications belonging to those data-flow classes.

Active Disks [11] is the seminal work regarding leveraging the aggregate processing power of CPU and memory in storage devices. The work aimed to achieve scalable data processing throughput through networked storage devices. The work may have failed to gain traction due to security concerns of downloading user software into storage devices which vendors would not like to give open access to. However, from our experience in demonstrating data transfers using DDN storage boxes at SC'14 [12], VM technology in disk controller provides safer environment from the perspective of vendors where privileged operations can be prohibited from being executed by VMs.

Recently, similar work by Kim *et al.* [13] investigated disk-layout aware data transfer in the context of high-performance computing. A major distinction of our work from [13] is that our method is lower-level approach in that our method can move data blocks of a given file set in any order while the method in [13] is still moving data on a file-by-file basis.

VIII. Conclusions and Future Work

We propose a serverless data movement architecture where both data transfer processes and parallel file system servers are embedded into a disk controller. This architecture can eliminate the communication overhead and potential network bottleneck between data transfer processes and parallel file system servers. The experimental results show that our novel architecture outperforms the existing architecture up to 103%. We also propose a data-layout-aware transfer method that can bypass a parallel file system layer after simply getting data block location information from the parallel file system. The data-layout-aware transfer method can further improve the performance up to 42%. We demonstrated our proposed embedded DTP architecture at SC'14 and achieved a peak throughput rate of 90 Gb between Starlight, Chicago and Ottawa, Canada [12]. In the future, we will develop more sophisticated data-layout-aware transfer algorithms for the more general problem between a sender with M disk controller between a receiver with N disk controller where data is distributed across multiple disk controllers. We also plan to deploy them in more high-speed networks and storage systems such as SSD storage, 40G NIC and terabit networks to prove that this method can push the limit of maximum data transfer rates.

Acknowledgments

The submitted manuscript has been created by UChicago Argonne, LLC, Operator of Argonne National Laboratory ("Argonne"). Argonne, a U.S. Department of Energy Office of Science laboratory, is operated under Contract No. DE-AC02-06CH11357. The U.S. Government retains for itself, and others acting on its behalf, a paid-up nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government.

This material was based upon work supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research Program, under Contract DE-AC02-06CH11357.

We gratefully acknowledge the computing resources provided and operated by the Joint Laboratory for System Evaluation (JLSE) at Argonne National Laboratory.

REFERENCES

[1] "Synergistic challenges in data-intensive science and exascale computing," http://science.energy.gov/~/media/40749FD92B58438594256267425C4AD1.ashx, Apr. 2014.

[2] "Science DMZ: A Scalable Network Design Model for Optimizing Science Data Transfers," `http://fasterdata.es.net/science-dmz/`.

[3] "DDN SFA12K platform," www.ddn.com/products/sfa12k.

[4] D. Arteaga and M. Zhao, "Towards Scalable Application Checkpointing with Parallel File System Delegation," in *2011 6th IEEE International Conference on Networking, Architecture and Storage (NAS)*, Jul. 2011, pp. 130–139.

[5] M. Rosenblum, *The Design and Implementation of a Log-Structured File System*, 1st ed. Norwell, MA, USA: Kluwer Academic Publishers, 1995.

[6] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, and I. Foster, "The globus striped GridFTP framework and server," in *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, ser. SC '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 54–64.

[7] T. Jones, A. Koniges, and R. Yates, "Performance of the IBM general parallel file system," in *Proceedings of the 14th International Parallel and Distributed Processing Symposium IPDPS*, 2000, pp. 673–681.

[8] E. Yildirim, J. Kim, and T. Kosar, "How GridFTP pipelining, parallelism and concurrency work: A guide for optimizing large dataset transfers," in *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*, 2012, pp. 506–515.

[9] "NDMP website," http://www.ndmp.org.

[10] "High performance storage system overview," http://www.hpss-collaboration.org/documents/HPSSIntroduction2009.pdf.

[11] E. Riedel, C. Faloutsos, G. Gibson, and D. Nagle, "Active disks for large-scale data processing," *Computer*, vol. 34, no. 6, pp. 68–74, Jun. 2001.

[12] "Pushing data transfer speed limit," http://www.ci.anl.gov/blog/pushing-data-transfer-speed-limit-sc14.

[13] Y. Kim, S. Atchley, G. R. Vallee, and G. M. Shipman, "LADS: Optimizing Data Transfers using Layout-Aware Data Scheduling," Feb. 2015.